# High level modeling of Partially Dynamically Reconfigurable FPGAs based on MDE and MARTE

Imran Rafiq Quadri, Samy Meftali and Jean-Luc Dekeyser,
INRIA LILLE NORD EUROPE - LIFL - University of Lille - CNRS, Lille, France
{Imran.Quadri, Samy.Meftali, Jean-Luc.Dekeyser}@lifl.fr

***Abstract*— System-on-Chip (SoC) architectures are becoming the preferred solution for implementing modern embedded systems. However their design complexity continues to augment due to the increase in integrated hardware resources requiring new design methodologies and tools. In this paper we present a novel SoC co-design methodology based on a Model Driven Engineering framework while utilizing the MARTE (Modeling and Analysis of Real-time and Embedded Systems) standard. This methodology permits us to model fine grain reconfigurable architectures such as FPGAs and allows to extend the standard for integrating new features such as Partial Dynamic Reconfiguration supported by modern FPGAs. The overall objective is to carry out modeling at a high abstraction level expressed in a graphical language like UML (Unified Modeling Language) and afterwards transformations of these models, automatically generate the necessary specifications required for FPGA implementation.**

## I. Introduction

Modern System-on-chips (SoCs) have become essential for designing embedded systems in order to target intensive parallel computation applications. While current SoC technological advances permit a rapidly increasing number of integrated transistors on a single chip in order to improve computational power, embedded system applications have also evolved becoming more sophisticated and resource demanding leading to a significant gap between design productivity and verification of these complex systems. An important challenge is to find efficient design methodologies that address the problems regarding these complex systems.

For the conception of a SoC, the behavioral description of the system is refined into an accurate register-transfer level (RTL) design usually by using High Level Synthesis (HLS) approaches. While an effective HLS flow has to be adaptable to rapidly evolving technologies and maintainable by the tool designers, in reality the abstraction level of the user-side tools is usually not elevated enough to be totally independent from low level implementations. Specifications usually are written in C/C++ or similar languages, leading to several disadvantages. First, one cannot differentiate between concepts easily as in a graphical representation. Second, system information related to e.g. hierarchy or data parallelism is not immediately visible and third, the modification process is complex and time consuming.

Model Driven Engineering [1] (MDE) is an emerging domain and can be seen as a *High Level Design Flow* and an effective solution for resolving the above mentioned problems. The advantage of MDE is that the complete system (both application and architecture) is modeled at a high specification level allowing several abstraction levels. A designer thus can focus on a particular domain space related to an abstraction level. The UML (Unified Modeling Language) graphical language allows to increase comprehensibility of the system and permits relations between concepts defined at different abstraction levels. High abstraction level descriptions of systems can be provided by the users and they can identify the internal concepts (task/data parallelism, data dependencies and hierarchy). The graphical nature of these specifications allows for their reuse, modification, maintenance and extension.

Partial Dynamic Reconfiguration [2] (PDR) is an emerging feature supported by modern FPGAs for reconfiguring specific portions of FPGA at run-time with the intent of time-sharing the available hardware resources for supporting multiple (mutually exclusive) tasks. Moreover, PDR permits swapping of tasks depending upon the application needs and Quality-Of-Service (QoS) requirements (performance, execution time etc.) and adds the possibility of developing future applications to target these adaptive architectures. Xilinx proposed the initial PDR methodology in [3] and currently only Xilinx FPGAs fully support this feature.

MARTE [4] (Modeling and Analysis of Real-Time and Embedded Systems) is an industry standard proposal of the Object Management Group (OMG) for model-driven development of embedded systems. It add capabilities to UML allowing to model software, hardware and their relations, along with added extensions (for e.g. performance and scheduling analysis). This standard although while rich in concepts, unfortunately lacks certain aspects for FPGA modeling.

GASPARD [5] is a MARTE compliant SoC co-design environment dedicated specially towards parallel hardware and software co-design allowing to move from high level MARTE specifications to an executable platform. It exploits the parallelism included in repetitive constructions of hardware elements or regular constructions such as application loops.

The main contribution of this paper is to present part of a novel design flow using an extended version of the MARTE standard for general modeling of FPGAs. This methodology also permits us to introduce PDR in MARTE for modeling all types of FPGAs supporting PDR. Finally by utilizing the MDE model transformations, the design flow can be used to bridge the gap between high abstraction levels and low implementation details to automatically generate the code required for the creation of bitstream(s) for FPGA implementation.

The rest of this paper is organized as follows. An overview of MDE is provided in section 2 while section 3 summarizes

our MARTE compliant GASPARD environment. Section 4 describes PDR while section 5 gives a summary of related works. Section 6 illustrates our methodology related to implementing PDR supported FPGAs. This paper finishes with a case study in section 7 followed by a conclusion.

## II. Model Driven Engineering

MDE is centered around three focal concepts. *Models*, *Metamodels* and *Transformations*. A model is an abstraction of reality and composed of concepts and relations. Concepts are "things" and relations are the "links" between these things in reality. A model can be observed from different point of views (views in MDE). A metamodel is in fact a collection of concepts and relations for describing a model. It defines the syntax of a model as a language defines its grammar. Each model is then said to *conform* to its metamodel.

A model transformation is a compilation process that transforms a *source* model into a *target* model and allows to move from an abstract model to a more detailed model. The source and target models each conform to their respective metamodels. A model transformation is based on a set of *rules* that help to identify concepts in a source metamodel in order to create enriched concepts in the target metamodel. This separation allows to easily extend and maintain the compilation process. New rules extend the compilation process and each rule can be independently modified. Model transformations carry out refinements moving from high abstraction levels to low levels for code generation. At each intermediate level, implementation details are added to the compilation process. The advantage of this approach is that it allows to define several model transformations from the same abstraction level but targeted to different lower levels, offering opportunities to generate several implementations from a specification.

## III. GASPARD Co-Design Environment

The GASPARD Environment [5] is based on a MDE approach for SoC co-design and is a subset of the MARTE standard that is currently supported by the industry. In MARTE, a clear separation exists between the hardware and the software components which is of prime importance in SoC conception. Our environment also uses the MARTE allocation mechanism (*Alloc* package) that permits to link the independent hardware and software models (for e.g. mapping of a task or data onto a processor or a memory respectively). The concept used to specify an allocation is called an *Allocate*. An allocation can represent either a *spatial* or a *temporal* placement. Up till now GASPARD only supported spatial placement but we have also integrated the temporal placement allocation in order to implement systems supporting PDR.

The last part on which GASPARD relies upon is the *Repetitive Structure Modeling (RSM)* annex. RSM has been inspired by the domain specific language known as ArrayOL [6] dedicated to intensive multidimensional signal processing. This package allows to describe the regularity of a system's structure (composed of repetitions of structural components interconnected in a regular connection pattern) and topology in a compact manner. GASPARD uses this package to model large regular hardware architectures (such as multiprocessor architectures) and parallel applications. We do not use any other MARTE packages due to the nature of the targeted applications which are *control and data flow oriented intensive signal processing (ISP) applications* (that are in the form of graph of tasks) within the broad domain of systems encompassed by MARTE. These applications are widely encountered in SoC design and introduce a Model of Computation (MoC) based on ArrayOL [6]. Although MARTE is generic enough to accommodate a large set of needs, the provided concepts lack information necessary for implementation purposes. Thus GASPARD introduces additional concepts and semantics to fill this requirement in the particular domain of SoC co-design.

The first addition relates to the semantics of modeled applications. In MARTE, nearly all kinds of applications can be specified but their behavior cannot be entirely defined. It is up to the designer/programmer to determine the precise behavior. As in GASPARD we deal with ISP applications based on a specific MoC [6], we only use the UML concept of *Component* (in order to define an application component) and MARTE *FlowPort* type (to define all port types in both the application and the architecture).

GASPARD also benefits from the notion of a *Deployment* model level [7] which is related to the specification of elementary components (basic building blocks of all other components). To transform the high abstraction levels to code, detailed information must be provided. The Deployment level links every elementary component to an existing code for both the hardware and the application. This facilitates Intellectual Property (IP) reuse. Each elementary component can have several implementations: e.g. an application functionality can either be optimized for a processor or written as an hardware accelerator. Hence this level is able to differentiate between hardware and software functionalities independent from the compilation target. It provides IP information for model transformations forming a compilation chain to transform the high abstraction level models (application, architecture and allocation) for different domains (formal verification, simulation, high performance computing or synthesis). This concept is currently not present in MARTE and is a potential extension for allowing a complete flow from model conception to automatic code generation. It should be noted that the different transformation chains (simulation, synthesis etc.) are currently unidirectional in nature.

## IV. Basic PDR related concepts

Currently PDR is only supported by Xilinx FPGAs. Xilinx initially proposed some methodologies [3],[8] followed by the *Early Access Partial Reconfiguration (EAPR)* [9] flow. The basic idea is that a part of the FPGA remains static, while another part is dynamically reconfigurable allowing the FPGA to be reconfigured at run-time. *Bus macros* are used to ensure proper routing between the static and dynamic parts during and after reconfiguration. Another important aspect is of the *Internal Reconfiguration Access Port (ICAP)* [10] that permits to read/write the FPGA configuration memory at run-time. The ICAP is present in nearly all Xilinx FPGAs ranging from

the low cost Spartan-3A(N) to the high performance Virtex-5 FPGAs [11]. For Virtex-II and Virtex-II Pro series, the ICAP furnishes an 8 bit input data bus and an 8 bit output data bus while with the Virtex-4 Series, the ICAP interface has been updated with 32 bit input and output data buses to increase its bandwidth. In combination with the ICAP, a *Reconfiguration controller* (either a PowerPC or a Microblaze) can be implemented inside the FPGA in order to build a self controlling dynamically reconfigurable system [10].

Virtex devices also support a feature of *glitchless dynamic reconfiguration*: If a configuration bit holds the same value before and after reconfiguration, the resource controlled by that bit does not experience any discontinuity in operation, with the exception of LUTRAMs and SRL16 primitives [2]. This limitation was removed in the Virtex-4 family. With the introduction of EAPR flow tools, this problem has also been resolved for Virtex-II/Pro FPGAs.

## V. Related Works

ROSES [12] is an environment for Multiprocessor SoC (MPSoC) design and specification but with a drawback as it does not conform to MDE concepts and as compared to our environment, starts from a low level description equivalent to our deployment level. While [13] provides a simulink based graphical HW/SW co-design approach for MPSoC, the MDE concepts are absent. In contrast, [14] uses the MDE approach for the design of a Software Defined Radio (SDR), but they do not utilize the MARTE standard as proposed by OMG. While works such as [15] and [16] are focused on generating VHDL from UML state machines, they fail to integrate the MDE concepts for HW/SW co-design and are not capable of managing ISP applications. MILAN [17] is another project for SoC co-design benefiting from the MDE concepts but is not MARTE compliant. Only the approach defined in [18] and [19] comes close to our intended methodology by using the MDE concepts and the MARTE standard for SoC co-design. Yet the disadvantage is that in reality it only generates the ISP application part to be implemented as a hardware accelerator in an FPGA. Hence there is no hardware description of FPGA at the high design level. MOPCOM [20] uses MDE and MARTE but is not oriented towards PDR.

In the domain related to PDR, Xilinx initially proposed two design flows in [3] and [8] termed as the *Modular based* and *Difference based* approaches. The difference based approach is suitable for small changes in a bitstream but is inappropriate for a large dynamically reconfigurable module necessitating the use of the modular approach. However, both approaches were not very effective leading to new alternatives.

Sedcole et al [21] presented a modular approach that was more effective than the initial Xilinx methodologies and were able to carry out 2D reconfiguration by placing hardware cores above each other. The layout (size and placement) of these cores is predetermined. They made use of reserved static routing in the reconfigurable modules which allowed the signals from the base region to pass through the reconfigurable modules allowing communication between modules by using the principle of glitchless dynamic reconfiguration.

Huebner et al [22] implemented 1D modular reconfiguration using a horizontal slice based bus macro. All the reconfigurable modules that stretched vertically to the height of the device were connected with the bus macro for communication. They followed by providing 2D placement of modules of any rectangular size by using routing primitives that stretch vertically throughout the device [23]. A module could be attached to the primitive at any location, hence providing arbitrary placement of modules. The routing primitives are LUT based and need to be reconfigured at the region where they connect to the modules. A drawback of this approach is that the number of signals passing through the primitives are limited due to the utilization of LUTs.

In 2006, Xilinx introduced the *Early Access Partial Reconfiguration (EAPR)* [9] flow along with the introduction of CLB based bus macros which are pre-routed IP cores. The concepts introduced in [21] and [22] were integrated in this flow. The restriction of full column modular PDR was removed allowing reconfigurable modules of any arbitrary rectangular size to be created. The EAPR flow also allows signals from the static region(s) to cross through the partially reconfigurable region(s) without the use of bus macros. Using the principle of glitchless reconfiguration, no glitches will occur in signal routes as long as they are implemented identically in every reconfigurable module for a region. The only limitation of this approach is that all the partial bitstreams for a module to be executed on a reconfigurable region must be predetermined.

Works such as [11] and [24] focus on implementing softcore internal configuration ports on Xilinx FPGAs such as the pure Spartan-3 which do not have the hardware ICAP core rendering dynamic reconfiguration impossible via traditional means. In [24] a soft ICAP known as JCAP (based on the serial JTAG interface) is introduced for realizing PDR while [11] introduces the notion of a PCAP (based on the parallel SelectMAP interface) providing improved reconfiguration rates as compared to the JTAG approach.

In [25], a new framework is introduced for implementing PDR by the utilization of a PLB ICAP. The ICAP is connected to the PLB bus as a master peripheral as compared to the traditional OPB based approach and provides an increased throughput of about 20 percent. [26] provides another flavor of a PDR architecture by attaching a Reconfigurable Hardware accelerator to a Microblaze Reconfiguration controller via a Fast Simplex Link (FSL) [27]. For our implementation purposes, we have focused mainly on the Xilinx EAPR flow methodology [2] as it is openly available and can be adapted to other PDR architecture implementations.

While there are lots of related tools, works and projects; we have only detailed some and have not given an exhaustive summary. To the best of our knowledge, only our methodology takes into account the four domain spaces: SoC HW/SW co-design, MDE, MARTE standard and PDR which is the novelty of our design flow.

## VI. Modeling of Partially Dynamically Reconfigurable FPGAs

We first present our design flow to model and implement PDR supported fine grain reconfigurable architectures (FP-

GAs) as shown in Figure.1 which is an extension of the design flow present in [19]. In this paper we only present the first model level of this flow (modeling of application, architecture and the allocation). Using model transformations, we will extend our work to link each modeled component with an IP at the Deployment model level (level 2). Following that, the RTL model level will provide detailed modeling information for the abstract concepts modeled at level 1 such as the reconfiguration controller and the reconfigurable hardware accelerator. Each of these model levels correspond to their respective metamodels. Finally, from the RTL level we will be able to automatically generate the specification for the Reconfiguration controller (for implementation in a processor) and the reconfigurable portion (level 4) for implementation in an FPGA using commercial tools. Our aim is not to replace the commercial tools but to aid them in the conception of a system. While tools like PlanAhead [28] are capable of estimating the FPGA resources required for a reconfigurable module, it is finally up to the user to decide the best placement depending on QoS requirements. Also as our work deals with dynamic partially reconfigurable FPGAs and currently only Xilinx FPGAs support this feature, our modeling methodology revolves around the Xilinx reconfiguration flow as it is openly available and flexible enough to be modified. While this does make the architectural aspects of our design flow restricted to Xilinx based technologies, it is an implementation choice as currently no other FPGA vendor supports this feature.

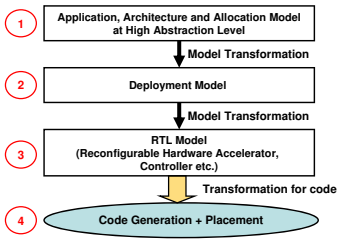


Fig. 1. *The complete design flow*

### A. Overview of MARTE Hardware concepts

In MARTE, The basic concepts of hardware are grouped in a package called *Hardware Resource Model (HRM)*. HRM is composed of two views, either a functional view (*HwLogical* sub-package), a physical view (*HwPhysical* sub-package) or a merge of the two. These two sub-packages derive from a root package called *HwGeneral* that revolves around the concept of a *HwResource* which defines a generic hardware entity. A HwResource can be composed of other HwResource(s) (for example a processor containing an ALU). This concept is then further enriched according to the functional or physical specifications. The functional view of HRM defines hardware resources as either *computing*, *storage*, *communication*, *timing* or *device* resources. The physical view represents hardware resources as physical components with details about their shape, size and power consumption among many other attributes. Until recently, our framework only supported the logical view but we have integrated both the physical and merged views in the framework for modeling PDR featured architectures. The HRM also exploits the Non-Functional Properties (NFP) package of MARTE. This package introduces an accurate value specification language for supporting complex expressions for specifying non-functional properties as well as quantitative annotations with measurement units. The NFP package provides a rich library of basic types like *Data size*, *Data Transmission Rate* and *Duration*.

### B. MARTE modifications for PDR concepts

In order to model PDR supported FPGAs, we examined the HRM package of MARTE and found it to be lacking in certain aspects. The *HwComputing* sub-package in the HRM functional view defines a set of active processing resources pivotal for an execution platform. A *HwComputingResource* symbolizes an active processing resource that can be specialized as either a processor (*HwProcessor*), an ASIC (*HwASIC*) or a PLD (*HwPLD*). An FPGA is represented by the HwPLD stereotype, it can contain a RAM memory (*HwRAM*) (as well as other HwResources) and is characterized by a technology (SRAM, Antifuse etc.). The cell organization of the FPGA is characterized by the number of rows and columns, but also by the type of architecture (Symmetrical array, row based etc.). These concepts are sufficient enough for FPGA description, however the concepts related to representing a processor are not sufficient for a complex SoC design in which a processor can either be implemented as a softcore IP or integrated as a hardcore IP. We thus add the attribute **imtype** (Implementation_Type) that is flexible enough to define a processor implementation as either **Hardcore** or **Softcore** and adaptable with future evolution using the **Other** and **Undefined** types. Figure.2 shows only the simplified modeling description of the modified HwComputing sub-package related to a processor implementation.

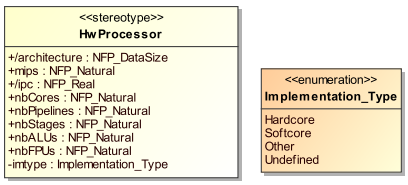


Fig. 2. *Modified version of the HwProcessor concept*

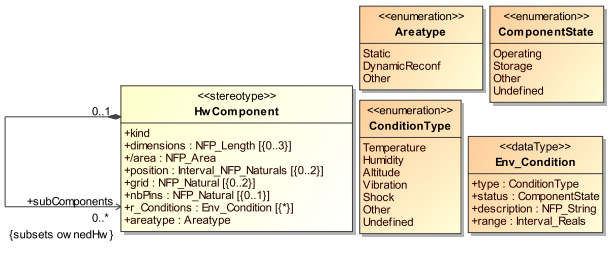


Fig. 3. *Modified version of the HwComponent concept*

In Figure.3 the second modification is shown which relates to the physical *HwLayout* sub-package that revolves around a concept of *HwComponent* which is an abstraction of any real

hardware entity based on its physical attributes. HwComponent can be specialized as either *HwChip* (e.g. a processor), *HwChannel* (e.g a bus), *HwPort* (e.g. an interface), *HwCard* (e.g. a motherboard) or a *HwUnit* (a hardware resource that does not fall into the preceding four categories). In order to specify the nature of the area for a PDR featured architecture (either static or dynamically reconfigurable), we have introduced the attribute **areatype** (Areatype) which can be either **Static**, **DynamicReconf** or typed as **Other** to adapt to future evolution. Although this concept can be implemented as a functional property, we have chosen to implement it in the physical view. Figure.3 thus shows only the simplified overview of our modified HwComponent concept.

These are the 2 general concepts that we have introduced at the conceptual level of the MARTE standard. These concepts are specifically added to the high level in order to generally benefit other frameworks and system descriptions and they could be easily extended. We now present the specific concepts related to FPGA and PDR in our methodology.

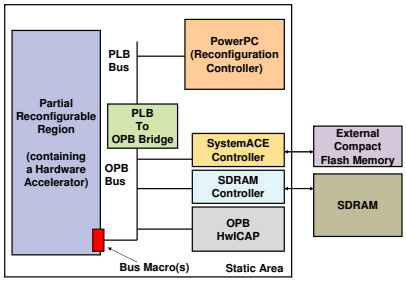


Fig. 4. *Block Diagram of the architecture of our reconfigurable system*

In Figure.4 we present a classical example of a PDR supported Xilinx FPGA. We have taken the Virtex-II Pro XC2VP30 on a XUP Board [29] as a reference as it seems to be a popular choice for implementing PDR. The architecture consists of a Reconfiguration Controller (a PowerPC in this case) connected to a 64-bit PLB bus and communicates with the slower slave peripherals (connected to the 32-bit OPB bus) via a PLB to OPB Bridge. The buses and the bridge are a part of the IBM Coreconnect technology [30]. The peripherals connected to the OPB bus are detailed as follows. A SystemACE controller for accessing the partial bitstreams placed in an external onboard Compact Flash (CF) card. A SDRAM controller for a DDR SDRAM present onboard that permits the partial bitstreams to be preloaded from the CF during initialization in order to decrease the reconfiguration time. An ICAP is present in the form of an OPB peripheral (OPBHwICAP) that permits partial reconfiguration using the read-modify-write mechanism [10]. The static portion of the FPGA is connected to a Reconfigurable Hardware Accelerator (RHA) via bus macros. Although we could have placed the RHA with the fast PLB bus, it is an implementation choice to connect it with the OPB bus. The concepts such as PowerPC, PLB and OPB buses, PLB to OPB Bridge, CF and SDRAM memories can be easily explained using the current MARTE HRM concepts. However the peripherals, bus macros, ICAP and RHA require an extended and more detailed conception. An internal memory can also be used to store the partial bitstreams depending upon the user requirements. For the moment, we have used an external SDRAM.

The *HwCommunication* sub-package in the HRM functional view represents the basic concepts for all hardware communications. *HwMedia* is the central concept defining a communication resource capable of data transfer with a theoretical bandwidth. It can be controlled by *HwArbiter*(s) and connected to other HwMedia(s) by means of a *HwBridge*. A *HwEndpoint* defines a connection point of a HwResource and can be defined as an interface (e.g. pin or port). *HwBus* illustrates a specific wired channel with particular functional attributes. These concepts are sufficient and abstract enough to define all kind of communication resources. Some of the other common HRM concepts that we utilize are *HwComputingResource* (to describe a general computing resource) from the *HwComputing* package, *HwRAM* and *HwROM* from the *HwMemory* package (for RAM and ROM concepts), *HwStorageManager* from the *HwStorageManager* package (for a memory controller), *HwClock* from the *HwTiming* package (to specify a clock) and *HwIO* from the *HwIO* package (for an I/O resource).

Xilinx provides the notion of an Intellectual Property Interface (IPIF) which is a hardware bus wrapper specially designed to ease IP core interfacing with the IBM Coreconnect buses. It can also be used for other purposes such as to connect the OPB bus to a DCR bus [30] (another bus of the Coreconnect technology). As all peripherals in our architecture consist of the IPIF wrapper and an IP core, this is a vital modeling concept and has permitted us to model all peripherals which are hierarchically composed. The IPIF has two basic attributes: a **mode** which can be either **Master**, **Slave** or **Master/Slave**, and **type** that determines the protocol of IPIF adapted for a particular bus. It can be either **PLB**, **OPB** or extensible using **Other** or **Undefined** types. We avoided adding detailed properties related to the protocols offered by IPIF to simplify its definition at the high abstraction level. The IPIF itself is typed *HwEndpoint* to denote that it is a hardware wrapper providing an interface to the IP core. This approach can be adapted to model customized wrappers for customized user IPs. Figure.5 shows the modeling of the IPIF.

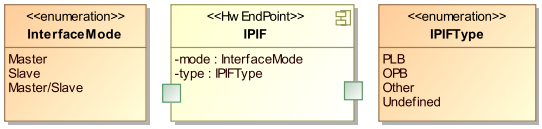


Fig. 5. *Modeling of the IPIF hardware wrapper*

The second modeling concept is that of Bus macros (BMs). Although the EAPR flow now allows signals in the base design to pass through the reconfigurable region(s) without the use of bus macros, they are still essential in order to ensure the correct routing between the static and dynamic regions. They are CLB based in nature and provide a unidirectional 8-bit data transfer. Bus macros have been modeled having four attributes. The **sigdir** attribute determines the direction of communication which can be **Left2Right** or **Right2Left** (for Virtex-II and Virtex-II Pro devices), as well as **Top2Bottom**,

**Bottom2Top** or **Other** for Virtex-IV and other future PDR supported devices. The **width** attribute determines the CLB width of the bus macro (**2CLBs** or **4CLBs** width making it either a narrow or wide bus macro or use of **Other** for a user specified width). The **Synchronous** attribute determines if the bus macro is synchronous or not. We have assigned a default value of **true** to this attribute (as recommended by Xilinx). The final attribute **device** determines the targeted FPGA device family (either **Virtex-II Pro**, **Virtex-II**, **Virtex-4** or a newer device such as Virtex-5 using the **Other** type). The Bus macro (*Busmacro*) as shown in Figure.6 is typed as *HwEndpoint* in order to illustrate that it is a communication medium between the static and dynamically reconfigurable modules of the FPGA.

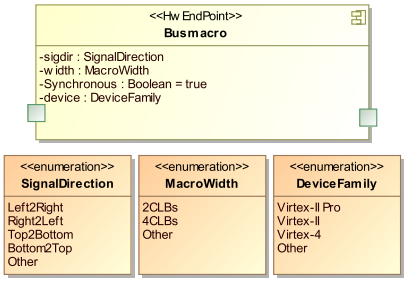


Fig. 6. *Modeling of a Bus macro*

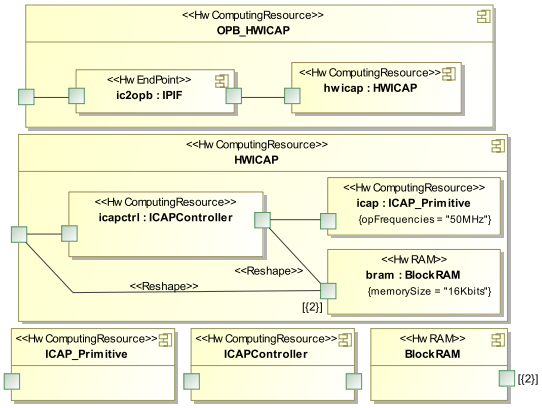


Fig. 7. *Modeling of the OPB HWICAP Peripheral*

We then carry out modeling of the OPB_HWICAP peripheral as shown in Figure.7. It consists of an IPIF (**ic2opb**) connected to the HWICAP core (**hwicap**) (typed as *HwComputingResource*) and is itself defined as a *HwComputingResource*. The HWICAP core is itself composed of three sub components: an ICAP controller (**icapctrl**) and ICAP Primitive (**icap**) both typed as *HwComputingResource*(s) and a BlockRAM (**bram**) defined as *HwRAM* for stocking a configuration frame of FPGA memory. The BlockRAM contains a port having a multiplicity of 2 indicating that it is repeated two times. We have used the notion of a *Reshape* connector (as defined in the MARTE RSM package) in order to link the sub components of the HWICAP. The Reshape allows to represent complex link topologies in a simplified manner. In Figure.7, the Reshape connectors permits to specify accurately which port (either the port of the ICAPController or the single port of the HWICAP itself) is connected to which repetition of the port of the BlockRAM. Also, the sub components of HWICAP have specific attributes (such as BlockRAM having a memory of 16Kbits) related to implementation details. We refer the reader to [10] for a detailed description of the HWICAP core.

Figure.8 illustrates the modeling of the Reconfigurable Hardware Accelerator (RHA). The PRR (Partial reconfigurable region) consists of a RHA (**HwAcc**) typed as *HwPLD* having ports **AccessOut** and **AccessIn** and an IPIF (**Acc2opb**). The PRR itself is of the generic *HwResource* type. The RHA is typed as *HwPLD* as it is reconfigurable, as compared to a typical hardware accelerator which can be seen as a *HwASIC* depending upon the designer's point of view.

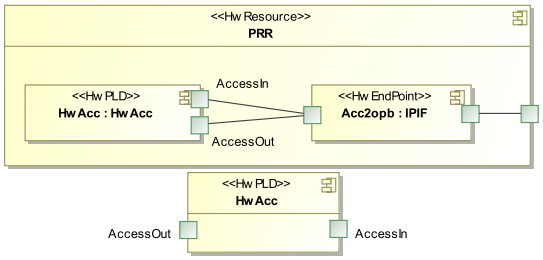


Fig. 8. *A Reconfigurable Hardware Accelerator*

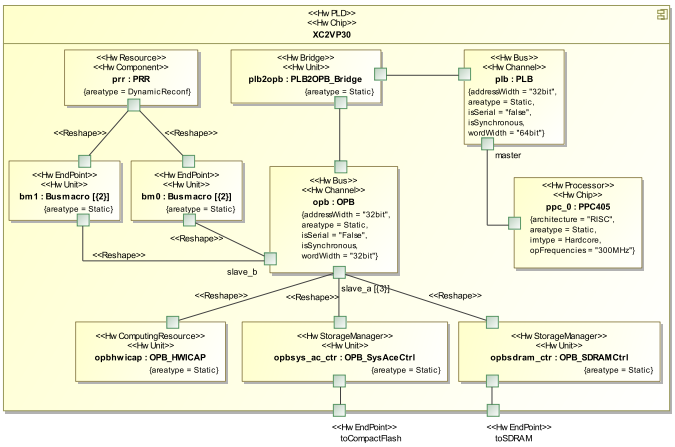


Fig. 9. *Modeling of our PDR Architecture*

Figure.9 finally illustrates our reconfigurable architecture (An XC2VP30 Virtex-II Pro chip) utilizing our proposed concepts in a merged functional/physical view. Each of the hardware components has two type definitions (the first representing the functional and the second representing the physical one). The XC2VP30 chip consists of a PowerPC PPC405 (**ppc_0**) connected via a PLB bus (**plb**) to the slave peripherals: the OPB_HWICAP (**opbhwicap**), the OPB_SysAceCtrl (**opbsys_ac_ctr**), the OPB_SDRAMCtrl (**opbsdram_ctr**) and the PRR (**prr**) via the OPB bus (**opb**). The PLB2OPB_Bridge (**plb2opb**) connects the two buses, while Bus macro(s) (**bm0** and **bm1** having types Left2Right and Right2Left respectively) connect the OPB bus to the PRR. Each of the bus macros is instantiated two times as indicated by the multiplicity of 2 on both **bm0** and **bm1** respectively. Also the OPB bus has a **slave_a** port with a multiplicity of 3 which allows the bus

to connect to the peripherals (opbsys_ac_ctr, opbsdram_ctr and opbhwicap), we have used Reshape connectors to determine which peripheral is connected to which repetition of the slave port. Similarly we have used Reshape connectors to determine the accurate connections between the bus macros and the ports of OPB and PRR. Although we could have used a single slave port on OPB with an appropriate multiplicity to include the topology of bus macros, this is avoided in order to reduce the design complexity. Finally, the XC2VP30 contains two HwEndPoint(s) interfaces, **toCompactFlash** and **toSDRAM** to connect **opbsys_ac_ctr** and **opbsdram_ctr** to the Compact Flash and the SDRAM memories respectively. Also, the OPB arbiter is not modeled as it is considered to be a part of the OPB Bus. It should be noted that this is a top level view only and nearly each component is itself hierarchically composed. Also the attributes introduced by us and those by default in the HRM package of MARTE allow the designer to specify general attributes of each component at the highest abstraction level (e.g. ppc_0 having a frequency of 300 MHz).

## VII. Case Study

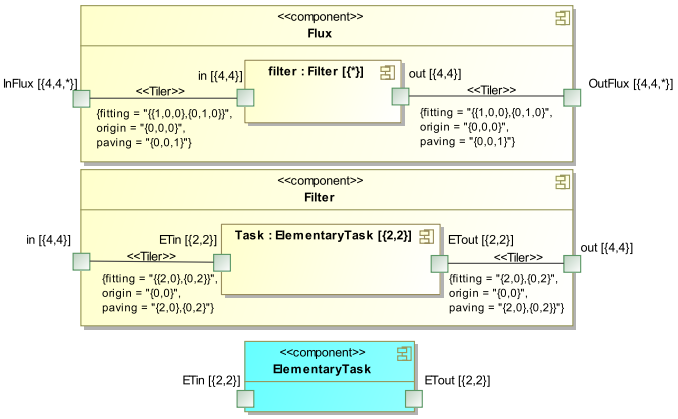


Fig. 10. *Model of an Image Filter task*

We provide here an example of a complete SoC model to validate our methodology and to give a concrete description of our usage of the MARTE standard. The modeled application *MainApplication* is an academic grayscale 4x4 pixel image filter application (producing 8-bit images). It consists of three application components: An image sensor PictureGen (**pg**), the main image filter task Flux (**tasks**) and finally an output PictureRead (**pr**). The Flux component is itself composed of a Filter component (**filter**) (repeated in an infinite dimension as shown by the multiplicity of *). The Filter component itself consists of an elementary application component Elementary-Task (**Task**) that is repeated four times (having a multiplicity of 2 by 2). The Tiler connectors are used to describe the tiling of produced and consumed arrays by a pattern mechanism [6]. The ElementaryTask can be associated with multiple IPs having different functionalities at the deployment level and the reconfiguration controller can choose several IPs for implementing PDR. The application components are not specifically typed as explained before in the paper. Figure.10 shows the image filter part of the application.

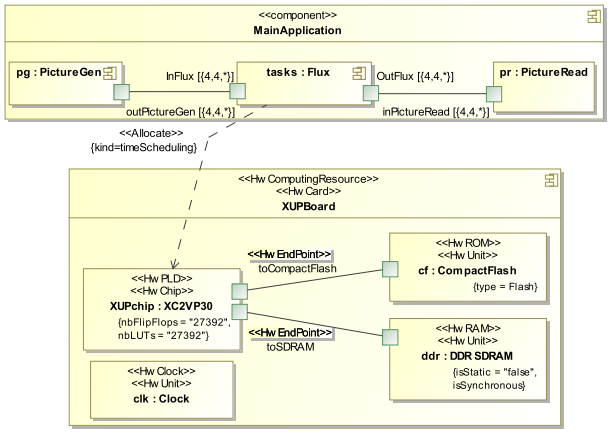


Fig. 11. *Allocation Level 1*

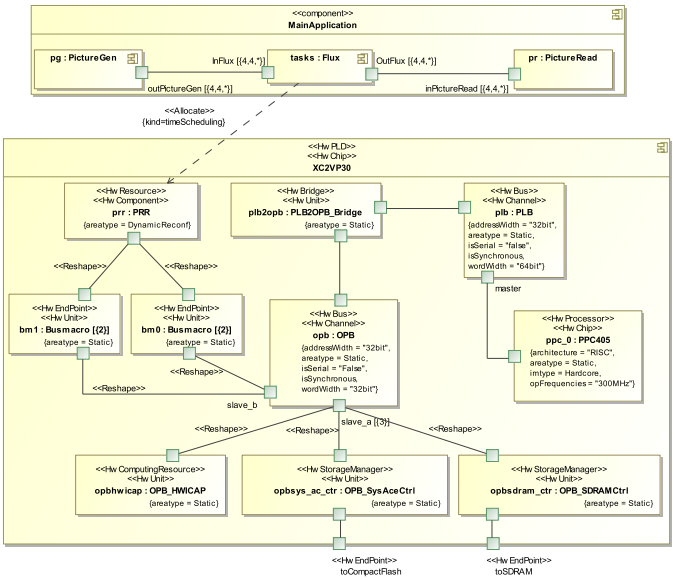


Fig. 12. *Allocation Level 2*

Figures.11 and 12 show the allocation of the application on to the architecture. In Figure.11, we show the model of the whole application with the image filter part allocated to the XC2VP30 chip (**XUPchip**) on an XUPBoard using the *Allocate* type allocation. Currently GASPARD only supports spacial placement (static scheduling at compilation time due to the nature of targeted applications), however due to the nature of PDR and related applications; we integrate the temporal placement: *timeScheduling* (dynamic scheduling of a set of elements spatially allocated to the same platform resource) nature of allocation as defined in MARTE. Figure.12 presents a detailed view of the allocation illustrating the mapping of the image filter task onto the PRR reconfigurable portion. Due to space limitations we have not presented the last level of allocation in which the image filter task is finally placed on a hardware accelerator **HwAcc**. The XUPBoard also contains a Clock (**clk**) and the CompactFlash (**cf**) and DDR SDRAM (**ddr**) memories. The concepts introduced in our approach can be modified and extended to manipulate other types of PDR supported architectures such as introduced in [25],

[26] and can be adapted to serve new emerging technologies such as explained in [11] and [24] validating our modeling approach. In order to validate our design methodology, we present another PDR architecture as shown in Figure.13. The figure shows the merged functional/physical modeling of a PLB ICAP based PDR architecture as defined in [25]. We have omitted some high level attribute specifications and type definitions in the figure in order to respect the space limitations of the paper. However, the modeling clearly illustrates that the PDR modeling methodology that we have proposed is easily extensible to include other PDR architectures and thus validates our claim.

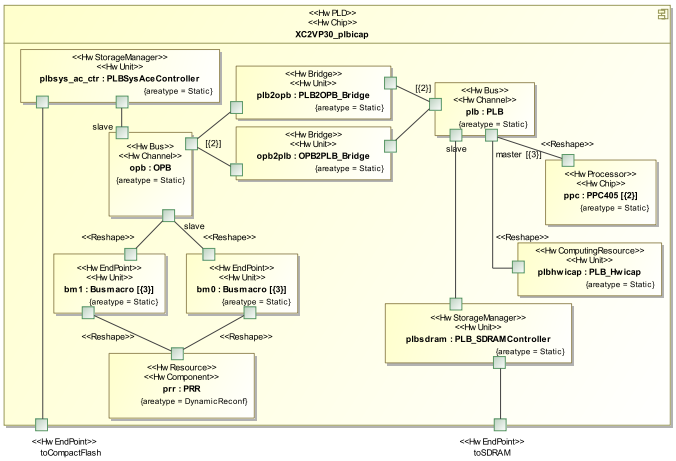


Fig. 13. *Modeling of a PLB ICAP based Reconfigurable Architecture*

Our modeling methodology can also be extended by integrating the *HwPhysical* arrangement notation that provides rectangular grid based placement mechanisms to make the UML diagrams appear closer to the real hardware topology. Due to current limitation of the modeling tools, it is not possible to model this view. However, this view could be a potential additional aid to commercial tools such as PlanAhead [28]. At the simulation level, designers can accurately estimate the FPGA resources consumed by reconfigurable modules by utilizing these tools and can modify their models at the high level resulting in a Design Space Exploration Strategy (DSE).

## VIII. Conclusions

In this paper, we have provided a new methodology to model PDR featured FPGAs based on a MDE approach using the MARTE standard. Initially, the MARTE standard has been modified to support general modeling of FPGAs by introduction of notions such as of peripherals and hardware wrappers. Afterwards, we have presented new concepts specifically for modeling and implementing PDR supported FPGAs. Our methodology is also extensible and can be adapted to serve other existing or future PDR based fine grain reconfigurable architectures. In future works, we will detail the model transformations and the low level Deployment and the enriched RTL models in order to automatically generate the necessary specifications required for FPGA implementation.

## References


[1] Planet MDE, "Portal of the Model Driven Engineering Community," 2007, http://www.planetmde.org.

[2] P. Lysaght et al, "Invited Paper: Enhanced Architectures, Design Methodologies and CAD Tools for Dynamic Reconfiguration of Xilinx FPGAs," in *FPL'06*, 2006.

[3] "Two Flows for Partial Reconfiguration: Module Based or Difference Based," in *Xilinx Application Note XAPP290, Version 1.1*, 2003.

[4] Object Management Group, "OMG MARTE Standard," 2007, http://www.omgmarte.org.

[5] The DaRT team, "GASPARD Design Environment," 2008, https://gforge.inria.fr/projects/gaspard2.

[6] P. Boulet, "Array-OL Revisited, Multidimensional Intensive Signal Processing Specification," INRIA, Tech. Rep., 2007, "http://hal.inria.fr/inria-00128840/en/".

[7] R.B. Atitallah et al, "Multilevel MPSoC simulation using an MDE approach," in *SoCC 2007*, 2007.

[8] "Two Flows for Partial Reconfiguration: Module Based or Difference Based," in *Xilinx Application Note XAPP290, Version 1.2*, 2004.

[9] Xilinx, "Early Access Partial Reconfigurable Flow," 2006, http://www.xilinx.com/support/prealounge/protected/index.htm.

[10] B. Blodget et al, "A lightweight approach for embedded reconfiguration of FPGAs," in *DATE'03*, 2003.

[11] S. Bayar and A. Yurdakul, "Dynamic Partial Self-Reconfiguration on Spartan-III FPGAs via a Parallel Configuration Access Port (PCAP)," in *2nd HiPEAC workshop on Reconfigurable Computing*, 2008.

[12] W. Cesario et al, "Component-Based Design Approach for Multicore SoCs," *DAC'02*, vol. 00, p. 789, 2002.

[13] Y.Atat and N. Zergainoh, "Simulink-based MPSoC Design: New Approach to Bridge the Gap between Algorithm and Architecture Design," in *ISVLSI'07*, 2007, pp. 9–14.

[14] G. Gailliard et al, "Transaction level modelling of SCA compliant software defined radio waveforms and platforms PIM/PSM," in *DATE'07*, 2007.

[15] R. Damasevicius and V. Stuikys, "Application of UML for hardware design based on design process model," in *ASP-DAC'04*, 2004.

[16] W.E. McUmber et al, "UML-based analysis of embedded systems using a mapping to VHDL," in *IEEE International Symposium on High Assurance Software Engineering (HASE'99)*, 1999, pp. 56–63.

[17] S. Mohanty et al, "Rapid design space exploration of heterogeneous embedded systems using symbolic search and multi-granular simulation," in *LCTES/Scopes 2002*, 2002.

[18] S. Le Beux et al, "A Model Driven Engineering Design Flow to generate VHDL," in *International ModEasy'07 Workshop*, 2007.

[19] S. Le Beux, "Un flot de conception pour applications de traitement du signal systématique implémentées sur FPGA à base d'Ingénierie Dirigée par les Modeles," Ph.D. dissertation, LIFL / USTL, France, 2007.

[20] A. Koudri et al, "Using MARTE in the MOPCOM SoC/SoPC Co-Methodology," in *MARTE Workshop at DATE'08*, 2008.

[21] P. Sedcole et al, "Modular Partial Reconfiguration in Virtex FPGAs," in *FPL'05*, 2005, pp. 211–216.

[22] J. Becker and M. Huebner and M. Ullmann, "Real-Time Dynamically Run-Time Reconfigurations for Power/Cost-optimized Virtex FPGA Realizations," in *VLSI'03*, 2003.

[23] M. Huebner et al, "New 2-Dimensional Partial Dynamic Reconfiguration Techniques for Real-Time Adaptive Microelectronic Circuits," in *ISVLSI'06*, 2006, p. 97.

[24] K. Paulsson et al, "Implementation of a Virtual Internal Configuration Access Port (JCAP) for Enabling Partial Self-Reconfiguration on Xilinx Spartan III FPGAs," *FPL 2007*, pp. 351–356, 2007.

[25] C. Claus et al, "A new framework to accelerate Virtex-II Pro dynamic partial self-reconfiguration," *IPDPS 2007*, pp. 1–7, 2007.

[26] A. Tumeo et al, "A Self-Reconfigurable Implementation of the JPEG Encoder," *ASAP 2007*, pp. 24–29, 2007.

[27] Xilinx, "Fast Simplex Link Channel (FSL)," 2004.

[28] N. Dorairaj et al, "PlanAhead Software as a Platform for Partial Reconfiguration," *Xcell Journal*, vol. 55, pp. 68–71, 2005.

[29] The Xilinx XUP-V2Pro Board, http://www.xilinx.com/univ/xupv2p.html.

[30] IBM Corporation, "The Coreconnect Bus Architecture, white paper," in *International Business Machines Corporation*, 2004.